Session 704

Compiler Technologies

John Graziano
Engineering Manager, Mac OS X
Compiler Group

Introduction

- Mac OS X Compiler Technologies
  - Compiler (gcc)
  - Assembler (as)
  - Linker (ld)
  - Dynamic Loader (dyld)
  - Utilities (strip, nm)
- Useable as command line tools
- Core technology for Project Builder

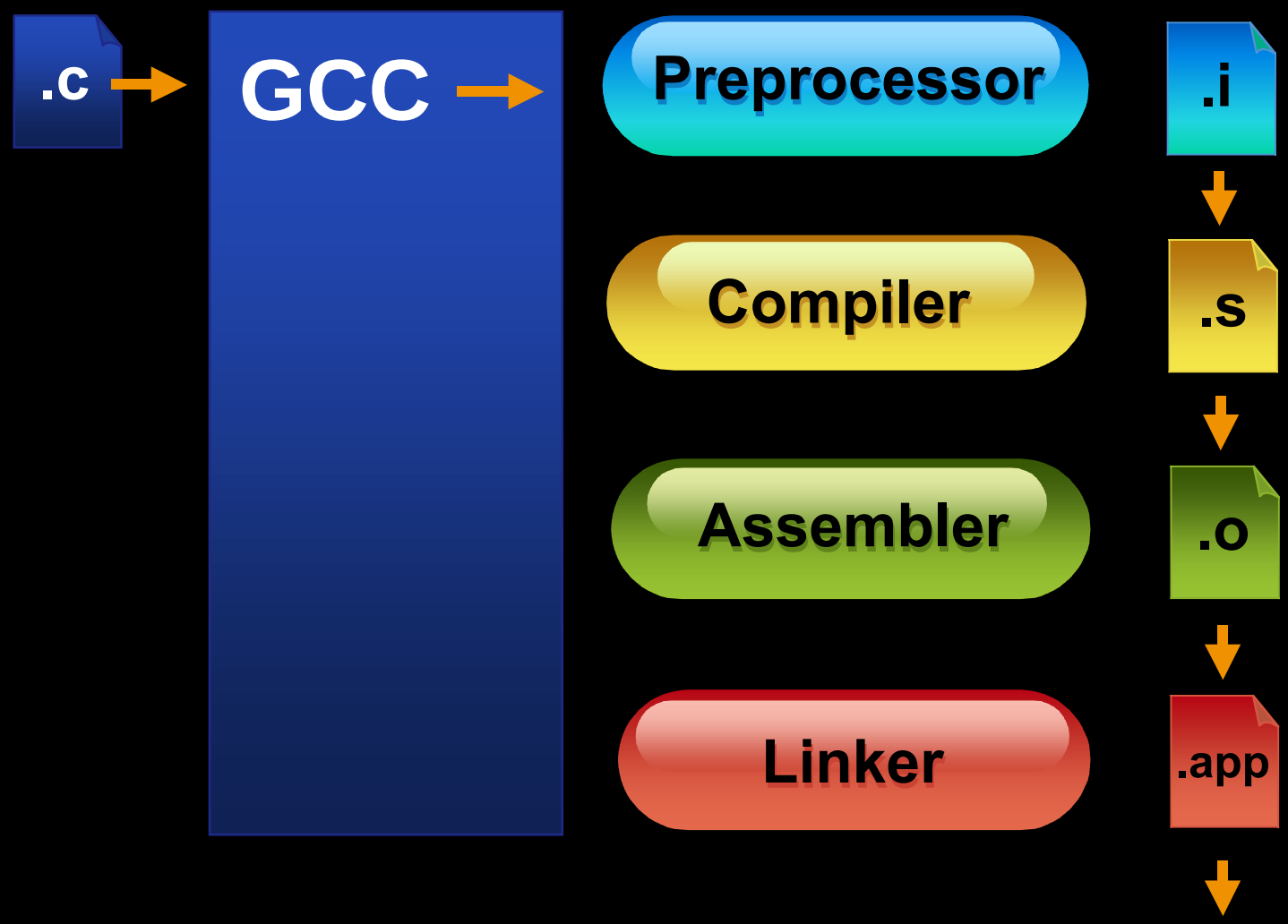
What We'll Cover

- Overview of GCC
- How to use GCC on Mac OS X
- Dynamic Linking and Prebinding
- Upcoming features

Gnu C Compiler

- Free Software
- Supports ANSI C, C++ and Objective-C
- Robust C++ implementation
- Many, many years of work and testing
- “Reference” compiler for many developers
- Compiles all of Mac OS X

GCC Architecture



Apple Extensions to GCC

- Project Builder Integration
- Objective-C
- Frameworks
- Precompiled Headers
- Velocity Engine (AltiVec)
- Mach-O

Differences Between GCC and Mac OS 9 Compilers

- Stricter ANSI C++ Compliance
  - Syntax
  - Type checking
- Static initializers
  - Order of execution may differ from Mac OS 9

New in Mac OS X GM

- Fully based on GCC 2.95.2
- C++ Improvements
  - Coalesced Symbols
  - Zero Overhead Exceptions
  - Precomps useable from C++

Coalesced Symbols

- Remove multiple copies of functions
  - Inline
  - Templates
- Finder binary reduced from 2Mb to 1.7Mb
- Reduction of overall system memory footprint

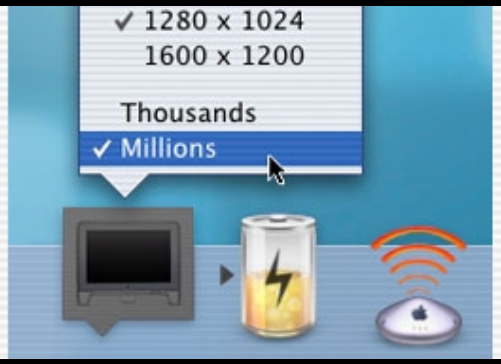
Zero Overhead Exceptions (ZOE)

- Removes performance cost for try blocks
- Common case (try block) very fast
  - 2x overall speedup in some real-world apps
- Finder size reduced an additional 400k
- Adds speed and memory penalty for throws
  - Throws cost 20x

Precompiled Headers With C++

- Compile times improved as much as 5x
- Precomps can be accessed from C++ but cannot contain C++ syntax
  - E.g., Carbon.h
- Macros and compiler flags must match

Using GCC on Mac OS X



Using GCC on Mac OS X

- Project Builder Integration
- Precompiled Headers
- Optimization
- AltiVec

GCC and Project Builder

- Project Builder is the preferred way to use GCC on Mac OS X
- Project Builder provides GUI for running GCC
  - Command line switches
  - Integrated error messages
- GCC provides source code indexing information to Project Builder

The Precompiler

- Creates a file of preprocessed and tokenized headers
- Selectively unparses declarations based on use
- Reduces memory footprint of compiler, reduces compiler parsing time

Using Precompiled Headers

- Already in major frameworks
  - Carbon, Cocoa
- Project Builder supports creating your own set of precompiled headers
- Preprocessor automatically finds precompiled headers

Precompiled Header Limitations

- Macro settings and timestamps must match
- Can be included from C++
- Cannot contain C++

Creating Your Own Precompiled Header

Devang Patel
Senior Compiler Engineer

Optimization

- Compiler optimization levels
- Controlling optimization from Project Builder

Optimization Level 0

- -O0 (or no flag) on command line
- No optimization performed
- Produces slower, bigger but debuggable code
- Gives fastest turnaround time
- Default setting for PBs development builds

Optimization Level 1

- -O or -O1 on command line
- Basic optimizations
  - Register allocation
  - Data flow analysis
  - Dead code elimination
  - Jump optimization
  - Peepholing
- Results in much smaller and faster code than level 0
  - Binary size reduces as much as 50%

Optimization Level 2

- Further function-level optimizations
  - Common sub-expression elimination
  - Loop optimization
  - Strength reduction
  - Instruction scheduling
- Produces faster (though not smaller) code than -O1
  - Some benchmarks improve 10–15%
- Code may be difficult to follow in debugger

Optimization Level 3

- Automatic function inlining
  - Can result in faster code
  - Binary size may increase
- Profiling is required, especially for C++
- Debugging code very difficult

Optimization Recommendations

- Use -O0 (none) for development builds
- Use -O2 for deployment builds
  - Reduces code size by about 50%
  - Increases execution speed as much as 2x
- Use -O3 only for cpu-critical cases
  - Carefully measure speed and size changes

AltiVec in GCC

- All AltiVec intrinsics supported in gcc
- Accepts same language extensions as CodeWarrior
- Requires -faltivec flag in Project Builder

Compiler Settings in Project Builder

Devang Patel
Senior Compiler Engineer

Dynamic Linking

- Binds references to functions and data at app startup time
- Function references allow lazy initialization of modules
- Data references require module initialization at startup

Prebinding

- Dynamic link addresses set at static link time
- Dramatic improvement in launch times
- No lazy initialization
- Applications are built prebound by default in Project Builder
- Binaries which are prebound must be updated when libraries change

Upcoming Features

- GCC 3
- Two-level namespace
- Objective-C++
- Further down the road…

GCC 3

- Integrated Preprocessor
- Better C++ Support
- Better codegen
- C++ ABI Changes!

Two-level Namespace

- Links by symbol and library name
- Protects against name collision between modules
- Speeds dynamic linking for non-prebound applications

Two-level Namespace Compatibility

- Mac OS X GM apps will work just fine on new systems
- Two-level apps will work just fine on GM systems
- Any combination of flat and two-level libraries will work as well

Objective-C++

Ziemowit (Zem) Laski
Senior Compiler Engineer, Mac OS X Compiler Group

Objective-C++ at a Glance

- Combines C++ and Objective-C
- Intended to allow C++ apps to utilize Objective-C frameworks (Cocoa!)
- New file extension: **.mm**
  - Legacy extension **.m** still supported (but discouraged)

ObjC++ Implementation

- Separate compiler front-end used for ObjC++
- Compiled ObjC++ programs interact with two runtimes
  - C++
  - Objective-C
- Separate and "mutually oblivious"
  - Syntax
  - Language runtimes

Mixing C++ and ObjC Code

- ObjC object instances (e.g., **id foo**, **NSObject *bar**) can be intermixed with C++ declarations
- ObjC classes may point to (and sometimes contain) C++ objects
- ObjC message sends (e.g., **[myObj foo]**) can be used in C++ expressions and vice versa
- ObjC++ uses C++ type semantics

ObjC++ Example

```
// HelloWorld.mm

#import <Cocoa/Cocoa.h>

class HelloWorld;

@interface PrintLog: NSObject {
  HelloWorld *ptr;
}
-  (void)sayHello;
-  (void)sayHi:  (HelloWorld *)p;
@end
```

ObjC++ Example (Cont.)

```
class HelloWorld {
  id printLog;
public:
  HelloWorld() { printLog = [[PrintLog alloc] init]; }
  void sayHi() { printf(”Hi”); }
  void sayHello() { [printLog sayHi: this]; }
};

@implementation PrintLog
-  (void) sayHello { NSLog(@"Hello, World!"); }
-  (void) sayHi: (HelloWorld *)p { p->sayHi(); }
@end
```

ObjC++ Restrictions

- Object hierarchies cannot mix

```
class Base { /* ... */ };

@interface ObjCClass: Base   // ERROR!
...
@end

class Derived: public ObjCClass   // ERROR!
{ /* ... */ };
```

ObjC++ Restrictions

- Should only use C++ (not ObjC) exceptions
- Must clean up ObjC objects yourself

```
id myObj;
try {
   myObj = [[MyClass alloc] init];
   ...
}
catch(...) {
   [myObj dealloc];
   throw;     // re-throw C++ exception
}
```

ObjC++ Restrictions

- C++ classes cannot receive ObjC messages (or vice versa)

```
CppClass *cppClass = new CppClass;
ObjCClass *objcClass = [[ObjCClass] alloc] init];

[cppClass objCMethod]; // ERROR!
objcClass->cppMethod(2);  // ERROR!
```

ObjC++ Restrictions

- Cannot statically allocate, **new** or **delete** ObjC objects

```
#import <Cocoa/Cocoa.h>

NSView myView;          // ERROR!

NSView *viewRef;        // ok -- but
                        // remember to alloc & init!

viewRef = new NSView;   // ERROR!
delete viewRef;         // ERROR!
```

ObjC++ Restrictions

- C++ classes with virtual member functions cannot be ObjC instance variables (ivars)

```
struct Class1 { virtual void foo(); }
struct Class2 { Class2(int i, int j); }

@interface ObjCClass {
   Class1 class1;   // ERROR!
   Class1 *ptr;     // ok
   Class2 class2;   // WARNING - constructor not called
}
...
@end
```

ObjC++ Availability

- Seeding coming soon!
- Looking for seed candidates now!

Send email, subject “ObjC++ Seeding” to:

Godfrey DiGiorgi
Technology Manager for Development Tools
Apple Worldwide Developer Relations
ramarren@apple.com

Still to Come…

- Compile time reductions
- Code quality improvements
- Tighter Project Builder integration

Summary

- GCC is road-tested, open compiler for Mac OS X
- Works from command line, works better from Project Builder
- Apple is fully committed to gcc as the main-line compiler for Mac OS X and Project Builder

Information Resources

Information pages for all Mac OS X tools
http://developer.apple.com/tools/

OpenSource at Apple web page
http://www.opensource.apple.com/

GNU Project web server
http://www.gnu.org/

Apple's mailing list master page
http://www.lists.apple.com/

Mailing lists:

projectbuilder-users	**projectbuilder-users**
cocoa-dev	**cocoa-dev**
carbon-dev	**carbon-dev**

Roadmap

705 Apple Performance Tools: Learn how to make your application scream!	Room A2 **Thurs., 5:00pm**
706 Debugging on Mac OS X: Debugging with Project Builder & GDB	Hall 2 **Fri., 9:00am**
707 Darwin Documentation Project & Header Doc	Room A2 **Fri., 10:30am**
FF119 Feedback Forum: Apple Developer Tools Give your feedback to the tools team!	Room J1 **Fri., 3:30pm**

Who to Contact

Godfrey DiGiorgi
Technology Manager for Development Tools
Apple Worldwide Developer Relations
ramarren@apple.com

macosx-tools-feedback@group.apple.com
Mailing address for feedback

http://developer.apple.com/events/urls.html

Q&A

Godfrey DiGiorgi
Technology Manager for Development Tools
ramarren@apple.com

http://developer.apple.com/events/urls.html

WWDC
Worldwide Developers Conference 2001